What is this?

What follows are additions to the CDF chapter of the *Scientific Data Formats* guide for IDL 6.3. The additions consist of:

- Modifications to the “Overview” section of the chapter describing the library number (upgraded from 2.7.1 to 3.1) and providing new links to CDF information.
  - Overview of the Common Data Format (Modified)
- New and modified routines. Modified routines show change bars in this document. New routines have no change bars. The new and modified routines are:
  - CDF_ATTGET (Modified)
  - CDF_CREATE (Modified)
  - CDF_ENCODE_EPOCH16
  - CDF_ENCODE_EPOCH (Modified)
  - CDF_EPOCH16
  - CDF_PARSE_EPOCH16
  - CDF_SET_CDF27_BACKWARD_COMPATIBLE
  - CDF_VARCREATE (Modified)

NOTE: keyword hyperlinks are not included in the “Syntax” sections of this version. The keyword links will be added in a later version.

(This page will not be included in the documentation.)

Overview of the Common Data Format

The Common Data Format is a file format that facilitates the storage and retrieval of multi-dimensional scientific data. This version of IDL supports version 3.1 of the CDF library. IDL's CDF routines all begin with the prefix "CDF_".

CDF is a product of the National Space Science Data Center (NSSDC). General information about CDF, including the "frequently-asked-questions" (FAQ) list, software, and CDF's IDL library (an alternative interface between CDF and IDL) are available on the World Wide Web at:

```
http://cdf.gsfc.nasa.gov/
```

CDF documentation, including the *CDF User's Guide*, is available at:

```
http://cdf.gsfc.nasa.gov/html/docs.html
```

For assistance via e-mail, send a message to the internet address:

```
cdfsupport@listserv.gsfc.nasa.gov
```

CDF_ATTGET

The CDF_ATTGET procedure reads an attribute entry from a CDF file.

Syntax

CDF_ATTGET, *Id*, *Attribute*, *EntryNum*, *Value* [, CDF_TYPE= *variable*] [, / ZVARIABLE]

Arguments

Id

The CDF ID, returned from a previous call to CDF_OPEN or CDF_CREATE.

Attribute

A string containing the name of the attribute or the attribute number to be written.

EntryNum

The entry number. If the attribute is variable in scope, this is either the name or number of the variable the attribute is to be associated with. If the attribute is global in scope, this is the actual gEntry. It is the user's responsibility to keep track of valid gEntry numbers. Normally, gEntry numbers will begin with 0 or 1 and will increase up to MAXGENTRY (as reported in the GET_ATTR_INFO structure returned by CDF_CONTROL), but this is not required.

Value

A named variable in which the value of the attribute is returned.

Keywords

CDF_TYPE

Set this keyword equal to a named variable that will contain the CDF type of the attribute entry, returned as a scalar string. Possible returned values are: CDF_CHAR, CDF_UCHAR, CDF_INT1, CDF_BYTE, CDF_UINT1, CDF_UINT2, CDF_INT2, CDF_UINT4, CDF_INT4, CDF_REAL4, CDF_FLOAT, CDF_REAL8, CDF_DOUBLE, CDF_EPOCH, or CDF_EPOCH16. If the type cannot be determined, "UNKNOWN" is returned.

Note that, as is true with variable data, attribute entries of type CDF_INT1, CDF_BYTE, CDF_UINT2, and CDF_UINT4 are converted into IDL supported datatypes (for example, data of type CDF_UINT2, data of the C-type unsigned short, is converted into IDL's INT, a signed integer. So, an attribute that is 65535 as a CDF_UINT2 will appear as INT = -1 in IDL). In these cases, pay special attention to the return values.

ZVARIABLE

If EntryNum is a variable ID (as opposed to a variable name) and the variable is a zVariable, set this flag to indicate that the variable ID is a zVariable ID. The default is to assume that EntryNum is an rVariable ID.

Note

The attribute must have a scope of VARIABLE_SCOPE.

Examples

```
; Open the CDF file created in the CDF_ATTPUT example:
id = CDF_OPEN('foo')
CDF_ATTGET, id, 'Att2', 'Var2', x
PRINT, X, FORMAT='("[",9(X,F3.1,","),X,F3.1,"]")'
CDF_CLOSE, id ; Close the CDF file.
```

IDL Output

```
[ 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```

This is the expected output, since this attribute was created with a call to FINDGEN.

Version History

Pre 4.0	Introduced
6.3	Add support for EPOCH_16 type

CDF_CREATE

The CDF_CREATE function creates a new Common Data Format file with the given filename and dimensions.

Note that when you create a CDF file, you may specify both encoding and decoding methods. Encoding specifies the method used to write data to the CDF file. Decoding specifies the method used to retrieve data from the CDF file and pass it to an application (IDL, for example). Encoding and decoding methods are specified by setting the *XXX*_ENCODING and *XXX*_DECODING keywords to CDF_CREATE. If no decoding method is specified, the decoding method is set to be the same as the encoding method.

All CDF encodings and decodings can be written or read on all platforms, but matching the encoding with the architecture used provides the best performance. Since most people work in a single-platform environment most of the time, HOST_ENCODING is the default encoding scheme for a new CDF file. If you know that the CDF file will be transported to a computer using another architecture, specify the encoding for the target architecture or specify NETWORK_ENCODING. Specifying the target architecture provides maximum performance on that architecture, specifying NETWORK_ENCODING provides maximum flexibility.

For more discussion on CDF encoding/decoding methods and combinations, see "Encoding" and "Decoding" in the *CDF User's Guide*.

Note

Versions of IDL beginning with 6.3 support the CDF 3.1 library. If you need to create CDF files that can be read by earlier versions of IDL, or by CDF libraries earlier than version 3.0, use the CDF_SET_CDF27_BACKWARD_COMPATIBLE routine.

Syntax

Result = CDF_CREATE(*Filename*, [*Dimensions*] [, /CLOBBER] [, /MULTI_FILE | , /SINGLE_FILE] [, /COL_MAJOR | , /ROW_MAJOR])

Encoding Keywords (pick one):

[, /ALPHAOSF1_ENCODING]
[, /ALPHAVMSD_ENCODING]
[, /ALPHAVMSG_ENCODING]
[, /DECSTATION_ENCODING]
[, /HOST_ENCODING]

[, /HP_ENCODING]
[, /IBMPC_ENCODING]
[, /IBMRS_ENCODING]
[, /MAC_ENCODING]
[, /NETWORK_ENCODING]
[, /NEXT_ENCODING]
[, /SGI_ENCODING]
[, /SUN_ENCODING]

Decoding Keywords (pick one):
[, /ALPHAOSF1_DECODING]
[, /ALPHAVMSD_DECODING]
[, /ALPHAVMSG_DECODING]
[, /DECSTATION_DECODING]
[, /HOST_DECODING]
[, /HP_DECODING]
[, /IBMPC_DECODING]
[, /IBMRS_DECODING]
[, /MAC_DECODING]
[, /NETWORK_DECODING]
[, /NEXT_DECODING]
[, /SGI_DECODING]
[, /SUN_DECODING]

Return Value

Returns the CDF ID for the new file.

Arguments

Filename

A scalar string containing the name of the file to be created. Note that if the desired filename has a `.cdf` ending, you can omit the extension and specify just the first part of the filename. For example, specifying `"mydata"` would open the file `mydata.cdf`.

Dimensions

A vector of values specifying size of each rVariable dimension. If no dimensions are specified, the file will contain a single scalar per record (*i.e.*, a 0-dimensional CDF). This argument has no effect on zVariables.

Keywords

CLOBBER

Set this keyword to erase the existing file (if the file already exists) before creating the new version.

Note that if the existing file has been corrupted, the CLOBBER operation may fail, causing IDL to display an error message. In this case you must manually delete the existing file from outside IDL.

COL_MAJOR

Set this keyword to use column major (IDL-like) array ordering for variable storage.

MULTI_FILE

Set this keyword to cause all CDF control information and attribute entry data to be placed in one `.cdf` file, with a separate file created for each defined variable. If the variable is an rVariable, then the variable files will have extensions of `.v0`, `.v1`, *etc.*; zVariables will be stored in files with extensions of `.z0`, `.z1`, *etc.* See “Format” in the *CDF User’s Guide* for more information. If both SINGLE_FILE and MULTI_FILE are set the file will be created in the SINGLE_FILE format.

Note

In versions of IDL prior to 6.3, MULTI_FILE was the default.

MULTI_FILE Example:

```
id=CDF_CREATE('multi', /MULTI_FILE)
CDF_CONTROL, id, GET_FORMAT=cdf_format
HELP, cdf_format
```

IDL prints:

```
CDF_FORMAT      STRING    = 'MULTI_FILE'
```

ROW_MAJOR

Set this keyword to specify row major (C-like) array ordering for variable storage. This is the default.

SINGLE_FILE

Set this keyword to cause all CDF information (control information, attribute entry data, variable data, etc.) to be written to a single `.cdf` file. This is the default. See

"Format" in the *CDF User's Guide* for more information. If both SINGLE_FILE and MULTI_FILE are set the file will be created in the SINGLE_FILE format.

Note

In versions of IDL prior to 6.3, MULTI_FILE was the default.

Encoding Keywords

Select one of the following keywords to specify the type of encoding:

ALPHAOSF1_ENCODING

Set this keyword to indicate DEC ALPHA/OSF1 data encoding.

ALPHAVMSD_ENCODING

Set this keyword to indicate DEC ALPHA/VMS data encoding using Digital's D_FLOAT representation.

ALPHAVMSG_ENCODING

Set this keyword to indicate DEC ALPHA/VMS data encoding using Digital's G_FLOAT representation.

DECSTATION_ENCODING

Set this keyword to select Decstation (MIPSEL) data encoding.

HOST_ENCODING

Set this keyword to select that the file will use native data encoding.

HP_ENCODING

Set this keyword to select HP 9000 data encoding. This is the default method.

IBMPC_ENCODING

Set this keyword to select IBM PC data encoding.

IBMRS_ENCODING

Set this keyword to select IBM RS/6000 series data encoding.

MAC_ENCODING

Set this keyword to select Macintosh data encoding.

NETWORK_ENCODING

Set this keyword to select network-transportable data encoding (XDR).

NEXT_ENCODING

Set this keyword to select NeXT data encoding.

SGI_ENCODING

Set this keyword to select SGI (MIPSEB) data encoding (Silicon Graphics Iris and Power series).

SUN_ENCODING

Set this keyword to select SUN data encoding.

Decoding Keywords

Select one of the following keywords to specify the type of decoding:

ALPHAOSF1_DECODING

Set this keyword to indicate DEC ALPHA/OSF1 data decoding.

ALPHAVMSD_DECODING

Set this keyword to indicate DEC ALPHA/VMS data decoding using Digital's D_FLOAT representation.

ALPHAVMSG_DECODING

Set this keyword to indicate DEC ALPHA/VMS data decoding using Digital's G_FLOAT representation.

DECSTATION_DECODING

Set this keyword to select Decstation (MIPSEL) data decoding.

HOST_DECODING

Set this keyword to select that the file will use native data decoding. This is the default method.

HP_DECODING

Set this keyword to select HP 9000 data decoding.

IBMPC_DECODING

Set this keyword to select IBM PC data decoding.

IBMRS_DECODING

Set this keyword to select IBM RS/6000 series data decoding.

MAC_DECODING

Set this keyword to select Macintosh data decoding.

NETWORK_DECODING

Set this keyword to select network-transportable data decoding (XDR).

NEXT_DECODING

Set this keyword to select NeXT data decoding.

SGI_DECODING

Set this keyword to select SGI (MIPSEB) data decoding (Silicon Graphics Iris and Power series).

SUN_DECODING

Set this keyword to select SUN data decoding.

Examples

Use the following command to create a 10-element by 20-element CDF using network encoding and Sun decoding:

```
id = CDF_CREATE('cdf_create.cdf', [10,20], /NETWORK_ENCODING, $
   /SUN_DECODING)
; ... other cdf commands ...
CDF_CLOSE, id ; close the file.
```

Now suppose that we decide to use HP_DECODING instead. We can use the CLOBBER keyword to delete the existing file when creating the new file:

```
id = CDF_CREATE('cdf_create.cdf', [10,20], /NETWORK_ENCODING, $
```

```
   /HP_DECODING, /CLOBBER)
; ... other cdf commands ...
CDF_CLOSE, id ; close the file.
```

The new file is written over the existing file. Use the following command to delete the file:

```
CDF_DELETE, id
```

Version History

Pre 4.0	Introduced
6.3	Changed default behavior to create a single file rather than multiple files (see SINGLE_FILE and MULTI_FILE)

CDF_ENCODE_EPOCH16

The CDF_ENCODE_EPOCH16 function encodes a CDF_EPOCH16 value into the standard date and time character string.

Syntax

Result = CDF_ENCODE_EPOCH16(*Epoch16*)

Return Value

Returns the string representation of the given CDF_EPOCH16 value.

Arguments

Epoch16

The double-precision CDF_EPOCH16 value to be encoded. This value can be

Keywords

EPOCH

Set this keyword equal to one of the following integer values, specifying the epoch mode to use for output of the epoch date string:

Value	Date Format
0	`DD-Mon-YYYY hh:mm:ss.ccc.uuu.nnn.ppp` (This is the default)
1	`YYYYMMDD.ttttttttttttt`
2	`YYYYMMDDss`
3	`YYYY-MM-DDThh:mm:ss.ccc.uuu.nnn.pppZ` (The characters `T` and `Z` are the CDF_EPOCH16 type 3 place holders)

where:

Date Element	Represents
DD	the day of the month (1-31)
Mon	the abbreviated month name: (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec)
MM	the month number (1-12)
YYYY	the year (A.D.)
hh	the hour (0-23)
mm	the minute (0-59)
ss	the second (0-59)
ccc	the millisecond (0-999)
uuu	the microsecond (0-999)
nnn	the nanosecond (0-999)
ppp	the picosecond (0-999)
tttttttttttttt	the fraction of the day (*e.g.* 500000000000000 is noon).

Examples

```
test_string = '04-Dec-2005 20:19:18.176.214.648.000'
test_epoch = CDF_PARSE_EPOCH16(test_string)
PRINT, CDF_ENCODE_EPOCH16(test_epoch)
```

IDL Output

```
04-Dec-2005 20:19:18.176.214.648.000
```

Version History

6.3	Introduced

See Also

CDF_PARSE_EPOCH16, CDF_EPOCH16

CDF_ENCODE_EPOCH

Note

This routine has been included in IDL for some time. It did not, however, appear in the online help *index* in prior to IDL 6.2. (This note will not be included in the documentation.)

The CDF_ENCODE_EPOCH function encodes a CDF_EPOCH variable into a string. Four different string formats are available. The default (EPOCH=0) is the standard CDF format, which may be parsed by the CDF_PARSED_EPOCH function or broken down with the CDF_EPOCH procedure.

Syntax

Result = CDF_ENCODE_EPOCH(*Epoch* [, EPOCH={0 | 1 | 2 | 3}])

Return Value

Returns a string containing the encoded CDF_EPOCH variable.

Arguments

Epoch

The double-precision CDF_EPOCH value to be encoded. For more information about CDF_EPOCH values, see “Data Types” in the *CDF User’s Guide*.

Keywords

EPOCH

Set this keyword equal to one of the following integer values, specifying the epoch mode to use for output of the epoch date string:

Value	Date Format
0	`DD-Mon-YYYY hh:mm:ss.ccc` (This is the default)
1	`YYYYMMDD.tttttt`

Value	Date Format
2	YYYYMMDDhhmmss
3	YYYY-MM-DDThh:mm:ss.cccZ (The characters T and Z are the CDF_EPOCH type 3 place holders)

where:

Date Element	Represents
DD	the day of the month (1-31)
Mon	the abbreviated month name: (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec)
MM	the month number (1-12)
YYYY	the year (A.D.)
hh	the hour (0-23)
mm	the minute (0-59)
ss	the second (0-59)
ccc	the millisecond (0-999)
ttttttt	the fraction of the day (*e.g.* 2500000 is 6 am).

Examples

```
epoch_string = '04-Dec-1995 20:19:18.176'
epoch = CDF_PARSE_EPOCH(epoch_string)
HELP, epoch_string, epoch

; Create encode strings:
encode0 = CDF_ENCODE_EPOCH(test_epoch, EPOCH=0)
encode1 = CDF_ENCODE_EPOCH(test_epoch, EPOCH=1)
encode2 = CDF_ENCODE_EPOCH(test_epoch, EPOCH=2)
encode3 = CDF_ENCODE_EPOCH(test_epoch, EPOCH=3)

; Compare encoding formats:
HELP, encode0, encode1, encode2, encode3
```

IDL Output

```
EPOCH_STRING    STRING    = '04-Dec-1995 20:19:18.176'
```

```
EPOCH           DOUBLE    =    6.2985328e+13

ENCODE0         STRING    = '04-Dec-1995 20:19:18.176'
ENCODE1         STRING    = '19951204.8467381'
ENCODE2         STRING    = '19951204201918'
ENCODE3         STRING    = '1995-12-04T20:19:18.176Z'
```

Version History

4.0.1b	Introduced

See Also

CDF_EPOCH, CDF_PARSE_EPOCH

CDF_EPOCH16 or CDF_EPOCH

The CDF_EPOCH16 or CDF_EPOCH procedure computes or parses an epoch value. When computing an epoch, any missing value is considered to be zero.

If you supply a value for the *Epoch* argument and set the BREAKDOWN_EPOCH keyword, the procedure will compute the values of the *Year*, *Month*, *Day*, *etc.* and insert the values into the named variables you supply.

If you specify the *Year* (and optionally, the *Month*, *Day*, *etc.*) and set the COMPUTE_EPOCH keyword, the procedure will compute the epoch and place the value in the named variable supplied as the *Epoch* parameter.

Note

You *must* set either the BREAKDOWN_EPOCH or COMPUTE_EPOCH keyword.

Syntax

CDF_EPOCH[16], *Epoch*, *Year* [, *Month*, *Day*, *Hour*, *Minute*, *Second*, *Milli*, *Micro*, *Nano*, *Pico*]

Arguments

Epoch

The Epoch value to be broken down, or a named variable that will contain the computed epoch will be placed. The Epoch value is the number of milliseconds or picoseconds since `01-Jan-0000 00:00:00.000`

Note

"Year zero" is a convention chosen by CDF to measure epoch values. This date is more commonly referred to as 1 BC. Remember that 1 BC was a leap year. The Epoch is defined as the number of picoseconds since `01-Jan-0000 00:00:00.000.000.000.000`, as computed using the CDF library's internal date routines. The CDF date/time calculations do not take into account the changes to the Gregorian calendar, and cannot be directly converted into Julian date/times. To convert between CDF epochs and date/times, use the CDF_EPOCH16 routine with either the BREAKDOWN_EPOCH or CONVERT_EPOCH keywords.

Year

If COMPUTE_EPOCH is set, a four-digit integer representing the year.

If BREAKDOWN_EPOCH is set, a named variable that will contain the year.

Month

If COMPUTE_EPOCH is set, an integer between 1 and 12 representing the month. Alternately, you can set *Month* equal to zero, in which case the *Day* argument can take on any value between 1-366.

If BREAKDOWN_EPOCH is set, a named variable that will contain the numeric month value.

Day

If COMPUTE_EPOCH is set, an integer between 1 and 31 representing the day of the month. Alternately, if the *Month* argument is set equal to zero, *Day* can be an integer between 1-366 representing the day of the year.

If BREAKDOWN_EPOCH is set, a named variable that will contain the numeric day of the month value.

Hour

If COMPUTE_EPOCH is set, an integer between 0 and 23 representing the hour of the day.

If BREAKDOWN_EPOCH is set, a named variable that will contain the numeric hour of the day value.

Minute

If COMPUTE_EPOCH is set, an integer between 0 and 59 representing the minute of the hour.

If BREAKDOWN_EPOCH is set, a named variable that will contain the numeric minute of the hour value.

Second

If COMPUTE_EPOCH is set, an integer between 0 and 59 representing the second of the minute.

If BREAKDOWN_EPOCH is set, a named variable that will contain the numeric second of the minute value.

Milli

If COMPUTE_EPOCH is set, an integer between 0 and 999 representing the millisecond.

If BREAKDOWN_EPOCH is set, a named variable that will contain the numeric millisecond value.

Micro

If COMPUTE_EPOCH is set, an integer between 0 and 999 representing the microsecond.

If BREAKDOWN_EPOCH is set, a named variable that will contain the numeric microsecond value.

Nano

If COMPUTE_EPOCH is set, an integer between 0 and 999 representing the nanosecond.

If BREAKDOWN_EPOCH is set, a named variable that will contain the numeric nanosecond value.

Pico

If COMPUTE_EPOCH is set, an integer between 0 and 999 representing the picosecond.

If BREAKDOWN_EPOCH is set, a named variable that will contain the numeric picosecond value.

Keywords

BREAKDOWN_EPOCH

Set this keyword to break down the value of the *Epoch* argument into its component parts, storing the resulting year, month, day, *etc*. values in the variables specified by the corresponding arguments.

COMPUTE_EPOCH

Set this keyword to compute the value of *Epoch* from the values specified by the *Year*, *Month*, *Day*, *etc*. arguments.

Examples

To compute the epoch value of September 20, 2005 at 3:05:46:02:156 am:

```
CDF_EPOCH16, epoch, 2005, 9, 20, 3, 5, 46, 27, 2, 156, $
   /COMPUTE_EPOCH
```

To break down the given epoch value into standard date components:

```
CDF_EPOCH16, epoch, yr, mo, dy, hr, min, sec, milli, micro, pico, $
   /BREAK
```

Version History

6.3	Introduced

CDF_PARSE_EPOCH16

The CDF_PARSE_EPOCH16 function parses a properly-formatted input string into a double-complex value properly formatted for use as a CDF_EPOCH16 variable.

Note

CDF_EPOCH16 variables may be unparsed into a variety of formats using the CDF_ENCODE_EPOCH16 or CDF_EPOCH16 functions.

Syntax

Result = CDF_PARSE_EPOCH16(*Epoch_string*)

Return Value

Returns the double-precision complex value of the input string properly formatted for use as a CDF_EPOCH16 variable.

Arguments

Epoch_string

A formatted string that will be parsed into a double precision complex value suitable to be used as a CDF_EPOCH value. The format of the date string is:

```
DD-Mon-YYYY hh:mm:ss.ccc.uuu.nnn.ppp
```

where:

Date Element	Represents
`DD`	the day of the month (1-31)
`Mon`	the abbreviated month name: (`Jan`, `Feb`, `Mar`, `Apr`, `May`, `Jun`, `Jul`, `Aug`, `Sep`, `Oct`, `Nov`, or `Dec`)
`YYYY`	the year (A.D.)
`hh`	the hour (0-23)
`mm`	the minute (0-59)
`ss`	the second (0-59)

Date Element	Represents
ccc	the millisecond (0-999)
uuu	the microsecond (0-999)
nnn	the nanosecond (0-999)
ppp	the picosecond (0-999)

Keywords

None

Example

```
test_string = '04-Dec-2005 20:19:18.176.214.648.000'
test_epoch = CDF_PARSE_EPOCH16(test_string)
CDF_EPOCH16, test_epoch, year, month, day, hour, min, sec, $
   milli, micro, nano, pico, /BREAKDOWN_EPOCH
HELP, test_string, test_epoch
PRINT, CDF_ENCODE_EPOCH16(test_epoch)
PRINT, year, month, day, hour, min, sec, milli, micro, nano, pico
```

IDL Prints:

```
TEST_STRING     STRING    = '04-Dec-1995 20:19:18.176'
TEST_EPOCH      DCOMPLEX  =    6.2985328e+13

04-Dec-2005 20:19:18.176.214.648.000
2005 12 4 20 19 18 176 214 648 000
```

Version History

6.3	Introduced

See Also

CDF_ENCODE_EPOCH16, CDF_EPOCH16

CDF_SET_CDF27_BACKWARD_COMPATIBLE

The CDF_SET_CDF27_BACKWARD_COMPATIBLE procedure allows users of IDL version 6.3 and later to create a CDF file that can be read by IDL 6.2 or earlier, or by CDF version 2.7.2 or earlier. By default, a CDF file created by IDL 6.3 or later cannot be read by earlier versions.

This procedure must be called prior to calling the CDF_CREATE function. It is useful if you need to create and share CDF files with colleagues who access CDF files using IDL version 6.2 or earlier, or using the CDF library version 2.7.2 or earlier.

In CDF version 2.7.2 and earlier, the maximum CDF file size was 2 Gbytes. This limitation was lifted in CDF version 3.0 with the use of a 64-bit file offset. As a result, users who use a CDF library older than CDF version 3.0 cannot read CDF files that were produced by CDF version 3.0 or a later release. CDF versions 3.0 and later *can* read files that were generated with any of the previous CDF releases.

Syntax

CDF_SET_CDF27_BACKWARD_COMPATIBLE [, /YES | /NO]

Arguments

None

Keywords

YES

Set this keyword to create a CDF file that can be read by IDL version 6.2 or earlier, or by CDF version 2.7.2 or earlier. If this keyword is set, the maximum file size is 2 Gbytes.

NO

Set this keyword to create a file that can only be read using IDL version 6.3 or later or CDF version 3.0 or later. This is the default when a CDF file is created.

Examples

Use the following command to create a CDF file that can be read by IDL 6.2 or earlier IDL versions.

```
CDF_SET_CDF27_BACKWARD_COMPATIBLE, /YES
id = CDF_CREATE('myfile.cdf')
CDF_CLOSE, id
```

Version History

6.3	Introduced

CDF_VARCREATE

The CDF_VARCREATE function creates a new variable in a Common Data Format file.

In CDF, variable is a generic name for an object that represents data. Data can be scalar (0-dimensional) or multi-dimensional (up to 10-dimensional). Data does not have any associated scientific context; it could represent an independent variable, a dependent variable, a time and date value, an image, the name of an XML file, etc. You can describe a variable's relationship to other variables via CDF's attributes.

CDF supports two types of variables: zVariables and rVariables. Different zVariables in a CDF data set can have different numbers of dimensions and different dimension sizes. All rVariables in a CDF data set must have the same number of dimensions and the same dimension sizes; this is much less efficient than the zVariable storage mechanism. (rVariables were included in the original version of CDF, and zVariables were added in a later version to address the rVariables' inefficient use of disk space.)

If you are working with a data set created using an early version of CDF, you may need to use rVariables. If you are creating a new CDF data set, you should use zVariables.

Syntax

```
Result = CDF_VARCREATE(Id, Name [, DimVary] [, VariableType]
                                    [ALLOCATERECS=records]
                                    [, DIMENSIONS=array]
                                    [, NUMELEM=characters]
                                    [, /REC_NOVARY | /REC_VARY]
                                    [, /ZVARIABLE] )
```

Return Value

Returns the variable of the type specified by the chosen keyword.

Arguments

Id
: The CDF ID, returned from a previous call to CDF_OPEN or CDF_CREATE.

Name
: A string containing the name of the variable to be created.

DimVary
: A one-dimensional array containing one element per CDF dimension. If the element is non-zero or the string `VARY', the variable will have variance in that dimension. If the element is zero or the string `NOVARY' then the variable will have no variance with that dimension. If the variable is zero-dimensional, this argument may be omitted.

Keywords

VariableType

You must specify the data type of the variable being created. This is done by setting one of the following keywords:

CDF_BYTE

CDF_CHAR

CDF_DOUBLE

CDF_EPOCH

CDF_LONG_EPOCH (creates a CDF_EPOCH16 variable)

CDF_FLOAT

CDF_INT1

CDF_INT2

CDF_INT4

CDF_REAL4

CDF_REAL8

CDF_UCHAR

CDF_UINT1

CDF_UINT2

CDF_UINT4

If no type is specified, CDF_FLOAT is assumed.

ALLOCATERECS

Set this keyword equal to the desired number of pre-allocated records for this variable in a SINGLE_FILE CDF file. Pre-allocating records ensures that variable data is stored contiguously in the CDF file. For discussion about allocating records, see “Records” in the *CDF User’s Guide*.

DIMENSIONS

Set this keyword to create a new zVariable with the specified dimensions. If

this keyword is not set, the variable is assumed to be a scalar. For example:

```
id = CDF_CREATE("cdffile.cdf", [100] )
zid = CDF_VARCREATE(id, "Zvar", [1,1,1],
DIM=[10,20,30])
```

Note
Variables created with the DIMENSIONS keyword set are always zVariables.

NUMELEM

The number of elements of the data type at each variable value. This keyword only has meaning for string data types (CDF_CHAR, CDF_UCHAR). This is the number of characters in the string. The default is 1.

REC_NOVARY

If this keyword is set, all records will contain the same information.

REC_VARY

If this keyword is set, all records will contain unique data. This is the default.

ZVARIABLE

Set this keyword to create a zVariable. For example:

```
id = CDF_CREATE("cdffile.cdf", [100] )
zid1 = CDF_VARCREATE(id, "Zvar1", /CDF_DOUBLE, /ZVARIABLE)
zid2 = CDF_VARCREATE(id, "Zvar2", /CDF_EPOCH, /ZVARIABLE)
zid3 = CDF_VARCREATE(id, "Zvar3", /CDF_LONG_EPOCH, /ZVARIABLE)
```

Note
zVariables are much more efficient than rVariables in their use of disk space. Unless the recipient of your data set is using a very early version of CDF, you should always create zVariables.

Note
Variables created with the DIMENSIONS keyword set are always zVariables.

Examples

Example 1

```
id = CDF_CREATE("temp_salinity.cdf)

temp_id =CDF_VARCREATE(id, "Temperature", ['Vary', 'Vary'], $
                       /CDF_FLOAT, /ZVARIABLE)
depth_id = CDF_VARCREATE(id, "Depth", [0,0], /REC_VARY, $
                         /CDF_FLOAT, /ZVARIABLE)
```

```
 ep1_id = CDF_VARCREATE(id, "epoch1", /CDF_EPOCH, /ZVARIABLE)
 ep2_id = CDF_VARCREATE(id, "epoch2", /CDF_LONG_EPOCH, /ZVARIABLE)

; Create and fill the UNITS attribute:
units_att = CDF_ATTCREATE(id, 'UNITS', /VARIABLE)
CDF_ATTPUT, id, 'UNITS', 'Depth', 'Meters'
CDF_ATTPUT, id, 'UNITS', temp_id, 'Kelvin'
CDF_ATTPUT, id, units_att, sal_id, 'Percent'

; Create and write some fictitous data:
data1 = 20.0 + FINDGEN(3,4)
CDF_VARPUT, id, varid, data1

; IDL will handle the type conversion, CDF will set all values
; of this record to a depth of 10.0.
CDF_VARPUT, id, depth_id, '10.0'
CDF_VARPUT, id, depth_id, 20.2, REC_START=1; Set the second depth.
CDF_VARPUT, id, sal_id, DINDGEN(3,4)/10.0

; Make more fictitous data.
; Demostrate the non-variance of depth by retrieving the
; values. On the first pass, use CDF_VARGET1 to retrieve
; single values:
CDF_VARGET1, id, depth_id, depth_0 ; Get single values.
CDF_VARGET1, id, depth_id, depth_1, REC_START=1

; Get single values.
HELP, depth_0, depth_1

; Now retrieve the full depth records:

CDF_VARGET, id, depth_id, depth, REC_COUNT=2
HELP, depth
PRINT, depth
```

IDL Output

```
DEPTH_0         FLOAT     =              10.0000
DEPTH_1         FLOAT     =              20.2000

 DEPTH          FLOAT     = Array(3, 4, 2)

10.0000        10.0000      10.0000
10.0000        10.0000
                            10.0000
10.0000        10.0000
                            10.0000
10.0000        10.0000
                            10.0000

20.2000        20.2000      20.2000
20.2000        20.2000
                            20.2000
20.2000        20.2000
                            20.2000
20.2000        20.2000
                            20.2000
```

Example 2

In this example, we create a variable, setting the data type from a string variable, which could have been returned by the DATATYPE keyword to a CDF_VARINQ call:

```
VARTYPE = 'CDF_FLOAT'
```

```
; Use the _EXTRA keyword and the CREATE_STRUCT function to
; make the appropriate keyword.

VarId = CDF_VARCREATE(Id, 'Pressure', [1,1], $

NUMELEM=2, _EXTRA=CREATE_STRUCT(VARTYPE,1))
CDF_CLOSE, id ; Close the CDF file.
```

Version History

Pre 4.0	Introduced
6.3	Add support for the CDF_EPOCH16 variable type

CDF_EPOCH_COMPARE

The CDF_EPOCH_COMPARE function compares two epoch (date and time) values and returns an integer value of 1, 0, or -1.

If the value of the first epoch is greater (later date and time) than the second epoch, 1 is returned. If the value of the first epoch is the same as the second epoch, 0 is returned. If the value of the first epoch is less (earlier date and time) than the second epoch, -1 is returned.

Syntax

Result = CDF_EPOCH_COMPARE (Epoch1, Epoch2)

Return Value

1 : Epoch1 > Epoch2 (Epoch1 is a later date and time than Epoch2)
0 : Epoch1 = Epoch2 (Epoch1 is the same as Epoch2)
-1 : Epoch1 < Epoch2 (Epoch1 is an earlier date and time than Epoch2)

Arguments

Epoch1
Epoch (date and time) value returned from CDF_VARGET, CDF_EPOCH, CDF_EPOCH16.

Epoch2
Epoch (date and time) value returned from CDF_VARGET, CDF_EPOCH, CDF_EPOCH16.

Examples

```
cdf_epoch, epoch1, 2005,6,1,10,18,17,2,/compute
cdf_epoch, epoch2, 2005,6,1,10,18,17,2,3,4,5,/compute

if cdf_epoch_compare(epoch1,epoch2) eq 0 then $
   print,"epoch1 = epoch2" $
else if cdf_epoch_compare(epoch1,epoch2) eq 1 then $
   print,"epoch1 > epoch2 – epoch1 is a later date than epoch2"
$
else $    ; return value is -1
   print,"epoch1 < epoch2 – epoch2 is a later date than epoch1"
```

Version History

Pre 4.0	Introduced
6.3	Add support for the CDF_EPOCH16 variable type